

For Reference

NOT TO BE TAKEN FROM THIS ROOM

Ex LIBRIS
UNIVERSITATIS
ALBERTAE NSIS



For Reference

NOT TO BE TAKEN FROM THIS ROOM

THE UNIVERSITY OF ALBERTA

A NUMERICAL ALGORITHM FOR DETERMINING
GRAPH ISOMORPHISM

by

Melvin L. Louie-Byne



A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE
OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

August, 1968



Digitized by the Internet Archive
in 2019 with funding from
University of Alberta Libraries

<https://archive.org/details/LouieByne1968>

THESIS
1968 (F)
135

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled A NUMERICAL ALGORITHM FOR DETERMINING GRAPH ISOMORPHISM submitted by Melvin L. Louie-Byne in partial fulfilment of the requirements for the degree of Master of Science.

ABSTRACT

This thesis presents an extension of the numerical algorithm, due to Julius and Charmonman, for determining graph isomorphisms for graphs with distinct eigenvalues to include graphs with one or more multiple eigenvalues. The extended algorithm is programmed and implemented on an IBM S/360/67 computer and an estimate of its efficiency is made.

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to my supervisor, Dr. S. Charmonman, for his guidance in the preparation of this thesis; to Dr. R.S. Julius, University of Toronto, for his advice in developing the algorithm; to Professor U.M. von Maydell for her interest and aid in proof reading this manuscript; to Miss S. Schultz, Secretary, for typing the thesis and her careful drawing of the graphs; and, to the Department of Computing Science, for providing the facilities and the financial assistance while this research was being performed. Also, I would like to thank the National Research Council of Canada for their financial assistance during the latter stages of this research.

TABLE OF CONTENTS

	Page
CHAPTER I - INTRODUCTION	
1.1 Literature Review	1
1.2 Purpose of the Study	6
CHAPTER II - BASIC THEORY	
2.1 Matrix Theory	7
2.1.1 Notation and Definitions	7
2.1.2 Matrix Outer Product	8
2.1.3 Kronecker Delta Function	9
2.1.4 Orthogonal Matrices	9
2.1.5 Permutation Matrices	10
2.1.6 Similar Matrices	10
2.2 Graph Theory	11
2.3 Special Operators	14
2.4 Graph Isomorphism Problem	15
CHAPTER III - ANALYSIS OF A NUMERICAL ALGORITHM TO DETERMINE GRAPH ISOMORPHISM	
3.1 Analysis of the Algorithm for Distinct Eigenvalues	17
3.1.1 Preliminary Analysis	17
3.1.2 Non-Singular Procedure	20
3.1.3 Singular Procedure	21
3.2 Analysis of the Algorithm for Multiple Eigenvalues	24
3.2.1 Preliminary Analysis	24
3.2.2 Procedure for Multiple Eigenvalues	41

	Page
CHAPTER IV - NUMERICAL EXPERIMENTS	
4.1 Test Data	44
4.1.1 Isomorphic Graphs with Distinct Eigenvalues	45
4.1.2 Isomorphic Graphs with Multiple Eigenvalues	47
4.1.3 Non-Isomorphic Graphs with Different Eigenvalues	49
4.1.4 Non-Isomorphic Graphs with Identical Eigenvalues	50
4.2 Summary of the Results	51
CHAPTER V - CONCLUSIONS AND SUGGESTIONS FOR FUTURE RESEARCH	
5.1 Conclusions	55
5.2 Suggestions for Future Research	57
BIBLIOGRAPHY	58
APPENDIX - LISTINGS OF FORTRAN IV SUBROUTINES, CONTROL PROGRAM AND NUMERICAL EXAMPLES	62

LIST OF FIGURES

	Page
4.1 The Complete 3-Node Graph	45

LIST OF TABLES

	Page
4.1 Isomorphic Graphs with Distinct Eigenvalues	46
4.2 Isomorphic Graphs with Multiple Eigenvalues	47
4.3 Non-Isomorphic Graphs with Different Eigenvalues	49
4.4 Non-Isomorphic Graphs with Identical Eigenvalues	50
4.5 Time Required to Determine that $G_i \approx G_i^*$	52
4.6 Time Required to Determine that $G_i \neq G_i^*$	53
4.7 Time Required to Generate all Permutation Matrices P Such that $A_i = PB_iP'$	54

CHAPTER I

INTRODUCTION

1.1 Literature Review

Since the appearance of the first paper on graph theory by Euler in 1736 [18], the theory of graphs has been developed extensively (see [3], [4], [13] and [19]) and is used in a wide variety of applications (see [3], [4], [6], [11], [13] and [18]). However, very little research was devoted to solving the graph isomorphism problem until the late 1950's. The research in this area has been primarily stimulated by the chemical industry in their attempts to develop an efficient chemical information retrieval system for the cataloguing and documenting of organic compounds ([1], [9], [10] and [28]). In a retrieval system, the problem of matching a given compound or query structure to those on file or in storage is equivalent to the graph isomorphism problem.

In theory, the graph isomorphism problem is, in a certain sense, trivial. That is, since there are only $n!$ ways in which the nodes of an n -node graph may be labelled, the graph isomorphism problem consists simply of generating the $n!$ mappings of one graph into the other and testing to see if the conditions for isomorphism are satisfied for any of these mappings. In practice, however, since $n!$

becomes extremely large even for relatively small n , this combinatorial approach is, in general, inefficient. For example, Unger [27] has estimated that it would require about one hour to compare two 10-node graphs and about forty years to compare two 15-node graphs using this approach. Moreover, these estimates were based on the modest assumption that each trial would require one millisecond.

The first attempts to solve the graph isomorphism problem were the node-by-node matching algorithms [5], [7] and [21]. These algorithms consist of matching two graphs node-by-node until either a complete match is found or until an incompatibility arises. When an incompatibility results, it is then necessary to backtrack to a point where there is agreement and try another path. This node-by-node matching technique requires an excessive amount of backtracking and if the two graphs are not isomorphic, then all possible paths must be tried before this conclusion is drawn. Moreover, Sussenguth [24] has shown that the time required to compare two n -node graphs by the node-by-node matching algorithms is an exponential function of n . Hence, the node-by-node matching technique is not efficient as a general algorithm for solving the graph isomorphism problem.

A heuristic algorithm (GIT) was developed by Unger [27]

for determining complete isomorphisms between directed line graphs. This algorithm was a significant improvement over the combinatorial methods and is described in detail in [27]. GIT was written in COMIT programming language and tested successfully on an IBM 7090 computer. The algorithm consists of setting up correspondences between sets of nodes of the two graphs that exhibit similar properties. The sets are generated according to certain nodal functions and the sets are then refined by various procedures. Occasionally, a guess is made with respect to specific nodes to see if an isomorphism results. Ultimately, the algorithm either detects an isomorphism between the two graphs or a contradiction is encountered, in which case no isomorphism is possible. In general, this algorithm is considerably more efficient than the previous ones and avoids a lot of unnecessary backtracking in most instances. However, if the graphs in consideration are highly symmetric and the nodal functions fail to distinguish the nodes to any great extent, then the algorithm has to distinguish nodes artificially and the amount of backtracking required by the algorithm increases considerably and hence the efficiency is reduced. Unger states that the graphs with a high degree of symmetry represent the most difficult cases for GIT. The time required to compare some non-trivial 7-node graphs was of the order of tens

of seconds and some 24-node graphs required about two minutes. A pair of 12-node graphs of the 'worst' type required approximately $7\frac{1}{2}$ minutes while a pair of 16-node graphs used 12 minutes of computing time without producing a solution. It was estimated that 12 additional minutes would have been required to produce a solution for the 16-node graphs.

Meanwhile, working independently at Harvard University, Sussenguth [24] developed a topological structure-matching procedure utilizing an approach similar to Unger's. The structure-matching algorithm of Sussenguth's is based on simple set and graph theoretic concepts. The algorithm sets up correspondences between sets of nodes of the two graphs which exhibit equivalent properties and uses a standard partitioning procedure, based on set theoretic principles, to reduce the sets into smaller subsets. Other procedures are used to generate new subsets until eventually, a one-to-one correspondence is delineated or a contradiction occurs, indicating that the two graphs are not isomorphic. The details of these procedures are described in [22], [23], [24] and [25]. The algorithm was programmed in assembly language and run successfully on an IBM 7090 computer. The algorithm developed by Sussenguth is more efficient than Unger's and has the additional feature that it is capable of detecting subgraph, partial graph and partial

subgraph matches. Moreover, Sussenguth has shown in [24] that the computation time required by the structure-matching procedure is proportional to n^2 . Furthermore, Sussenguth's structure-matching algorithm requires little or no backtracking unless the degree of symmetry of the graphs is high. However, for graphs with up to fifty nodes, isomorphisms were detected in 6 to 7 seconds while only a few milliseconds were required to determine that no isomorphism existed.

Several authors have considered the graph isomorphism problem as an eigenvalue problem including Harary [12], Julius and Charmonman [14] and Turner [26]. In September 1960, it was conjectured by Harary [12] that two graphs are isomorphic if their adjacency matrices have the same eigenvalue spectra. This conjecture was proved false by counter-example and hence has been withdrawn. However, this conjecture has been verified by exhaustive procedures for graphs with up to 6 nodes (see [12]).

In a recent paper by Turner [26], an attempt was made to characterize graphs up to isomorphism by considering a 'more powerful' set of matrix functions called immanents [15] or generalized matrix functions [16]. However, the efforts in this direction failed and two examples are given of non-isomorphic graphs whose adjacency matrices have the same generalized characteristic polynomial [26].

In [14], Julius and Charmonman have presented a solution to the complete graph isomorphism problem for undirected graphs which have distinct eigenvalues. The procedure is unique in that it depends only on the eigenvalues and eigenvectors of the adjacency matrices. Moreover, the algorithm requires no backtracking or guessing and is the only numerical algorithm encountered in the literature survey. A complete analysis of the algorithm is included in this thesis in Section 3.1 of Chapter III. Unfortunately, this algorithm has not been programmed and hence no estimate of the efficiency is available.

1.2 Purpose of the Study

The purpose of this research is to extend the numerical algorithm of Julius and Charmonman to include graphs which have one or more multiple eigenvalues and to program and implement the extended algorithm on an IBM S/360/67 computer in order to determine its efficiency.

CHAPTER II

BASIC THEORY

2.1 Matrix Theory

In this section, we discuss some of the basic definitions and theorems of matrices which are used throughout this paper. The theorems are stated without proof as the proofs can be found in most standard texts on Linear Algebra or Numerical Analysis [8], [17], and [20].

2.1.1 Notation and Definitions Unless otherwise specified, upper-case Latin and Greek letters denote matrices and lower-case letters denote scalars. We say that a rectangular matrix $A = (a_{ij})$ with m rows and n columns is of dimension m -by- n . If $m = n$, then A is square and of order n . The identity matrix of order n is designated by I unless there is ambiguity in which case, it is designated by I_n . The transpose of an arbitrary matrix A is denoted by A' and the inverse of an arbitrary non-singular matrix A is denoted by A^{-1} .

Given a matrix A of dimension m -by- n , the i^{th} row is designated by $R_i(A)$ and the j^{th} column by $C_j(A)$. The transpose of the i^{th} row is denoted by $R_i'(A)$ and similarly, the transpose of the j^{th} column is denoted by $C_j'(A)$. $R'(A)$ is a column vector of mn elements, given by

$$(2.1) \quad R'(A) = \begin{bmatrix} R'_1(A) \\ R'_2(A) \\ \vdots \\ R'_m(A) \end{bmatrix},$$

and $C'(A)$ is a row vector of mn elements, given by

$$(2.2) \quad C'(A) = [C'_1(A), C'_2(A), \dots, C'_n(A)] .$$

2.1.2 Matrix Outer Product The matrix outer product or Kronecker product (Bellman [2]) is denoted by the symbol \otimes and is defined as follows:

Definition 2.1. Given two matrices A and B of dimension m -by- n and p -by- q , respectively, the outer product, $A \otimes B$, is of dimension mp -by- nq and is given by

$$(2.3) \quad A \otimes B = \begin{bmatrix} a_{11}^B & a_{12}^B & \dots & a_{1n}^B \\ a_{21}^B & a_{22}^B & \dots & a_{2n}^B \\ \cdot & \cdot & \cdot & \cdot \\ a_{m1}^B & a_{m2}^B & \dots & a_{mn}^B \end{bmatrix} .$$

Theorem 2.1. If A, B, C and D are matrices such that the matrix equation

$$(2.4) \quad A = BCD$$

is valid, then by utilizing the matrix outer product, we can rewrite (2.4) as a system of simultaneous linear equations as follows:

$$(2.5) \quad R'(A) = [B \otimes D']R'(C) .$$

2.1.3 Kronecker Delta Function

Definition 2.2. The Kronecker delta function is denoted by δ_{ij} and is equal to one if $i=j$ and is zero otherwise.

2.1.4 Orthogonal Matrices

Definition 2.3. A real matrix Q of order n is called orthogonal if

$$(2.6) \quad Q' = Q^{-1}$$

Theorem 2.2. If A is a real symmetric matrix, then its matrix of eigenvectors is orthogonal.

2.1.5 Permutation Matrices A permutation matrix (Wilkinson [29]) is defined as follows:

Definition 2.4. A matrix $P = P_{\alpha_1, \alpha_2, \dots, \alpha_n}$ is a permutation matrix if

$$(2.7) \quad p_{ij} = \begin{cases} 1, & j = \alpha_i \\ 0 & \text{otherwise,} \end{cases}$$

where $(\alpha_1, \alpha_2, \dots, \alpha_n)$ is some permutation of $(1, 2, \dots, n)$.

A permutation matrix has the property that every row and column has precisely one unit element. Furthermore, if P is a permutation matrix, then P' is a permutation matrix. Moreover, P is orthogonal, that is,

$$(2.8) \quad P' = P^{-1}$$

and $P'P = PP' = I$. Further properties of P are that pre-multiplication of a matrix A by P replaces $R_i(A)$ by $R_{\alpha_i}(A)$ and post-multiplication by P' replaces $C_i(A)$ by $C_{\alpha_i}(A)$.

2.1.6 Similar Matrices (Murdoch [17])

Definition 2.5. Two matrices A and B are

similar if there exists a non-singular matrix S such that

$$(2.9) \quad B = S^{-1}AS .$$

(2.9) is a similarity transformation and $S^{-1}AS$ is called the transform of A by S .

Theorem 2.3. Similar matrices have the same eigenvalues. That is, if U and V are the eigenvectors of A and B respectively, then

$$(2.10) \quad AU = U\Lambda$$

and

$$(2.11) \quad BV = V\Lambda ,$$

where Λ is the diagonal matrix of eigenvalues.

2.2 Graph Theory

Because the terminology and symbolism pertaining to graph theory has not yet been standardized, this section is devoted to defining the terms and symbols which are used throughout this paper. The majority of the definitions and notations are taken from Busacker and Saaty [4].

Since geometric graphs may be considered as convenient representations of all graphs [4], a graph is defined as follows:

Definition 2.6. A graph consists of a non-null set of points (nodes, vertices) V in E^m , interconnected by a set of simple curves (edges, branches) E , which may be empty.

A graph is referred to as $G = (V, E)$ or simply G and if the set V has n elements, $n > 0$, then G is an n -node graph.

An undirected graph (graph) is one in which the edges have no associated direction or orientation. A directed graph (digraph) is a graph where each edge has a specific orientation. In this case, the edges will be referred to as arcs. A digraph will be denoted by $D = (V, A)$ and for every D , there is an associated undirected graph $G = (V, A)$ which is obtained from D by ignoring the orientation of the arcs.

If e is an edge with end-points v and w , then e is said to be incident with nodes v and w and the nodes v and w are incident with edge e . Two vertices are said to be adjacent if they are joined by an edge (or arc) e . Similarly, e_1 and e_2 are adjacent edges (or arcs) if they have at least one common end-point. A loop

is an edge e with end-points v and w where $v = w$. Two edges in a graph are said to be parallel if and only if they have the same end-points. In a digraph, two arcs are parallel if they have the same end-points but opposite orientation and are strictly parallel if the orientation is the same.

A simple graph is a graph that contains no loops and no parallel edges; and, a complete graph is one in which every pair of distinct vertices are adjacent.

We can formally define isomorphism between graphs as follows:

Definition 2.7. Two graphs, $G = (V, E)$ and $G^* = (V^*, E^*)$, are isomorphic to each other if there exists a one-to-one correspondence, Φ , between V and V^* and between E and E^* that preserves incidences.

That is, if $\Phi(v) = v^*$ and $\Phi(e) = e^*$ then edge e is incident with vertex v in G if and only if e^* is incident with vertex v^* in G^* . If G is isomorphic to G^* , we will write $G \approx G^*$, otherwise, $G \not\approx G^*$. Two digraphs are said to be isomorphic if their associated undirected graphs are isomorphic and each pair of corresponding arcs are oriented the same way.

A graph or digraph is completely specified by its adjacency (connection, vertex) matrix which is defined as follows:

Definition 2.8. Let the matrix A of order n be associated with the n -node graph (digraph) G . If the rows and columns of A correspond to the nodes of G , then define a_{ij} to be equal to the number of edges (arcs) joining the nodes i and (to) j . The matrix A is said to be the adjacency matrix of graph G .

It should be noted that if G is simple, then the elements $a_{ij} = 1$ or 0 for all i and j . Also, if a digraph D has no parallel or strictly parallel arcs and if A is the adjacency matrix of D , then the adjacency matrix of the associated undirected graph is given by the Boolean sum of A and A' .

2.3 Special Operators

In this section, we define two special operators which are used in the development of the graph isomorphism algorithms in Chapter III.

Definition 2.9. The superscript $+$ will be used instead of the standard division operator when the denominator can assume all real values including zero. The operator is defined as follows:

$$(2.12) \quad a^+ = \begin{cases} \frac{1}{a}, & a \neq 0 \\ 0, & a = 0 \end{cases}.$$

Definition 2.10. The binary operator \sim is to be interpreted to mean 'does not disagree with' and $0 \sim \pm a$, $+a \sim +a$, $-a \sim -a$ but $+a \not\sim -a$, where a is any real number. When this operator is used with vectors as arguments, we apply the operator to the corresponding elements in each vector.

2.4 Graph Isomorphism Problem

In this section, we formulate the graph isomorphism problem as an eigenvalue problem.

Since the nodes of an n -node graph may be labelled in $n!$ different ways, an unlabelled n -node graph G may be represented by up to $n!$ different adjacency matrices. If we associate the integers $1, 2, \dots, n$ with the nodes of G , then the $n!$ ways of labelling G correspond to the $n!$ permutations, $(\alpha_1, \alpha_2, \dots, \alpha_n)$, of the integers $1, 2, \dots, n$. It follows from Definition 2.7, that two graphs, G and G^* , are isomorphic if and only if they are differently labelled versions of the same graph. Furthermore, since graphs are completely specified by their adjacency matrices and if A and B are the respective adjacency matrices of G and G^* , then the problem of determining whether or not $G \sim G^*$ is essentially that of determining whether or not A and B represent differently labelled versions of the same graph.

If $G \approx G^*$, then there exists a permutation, $(\alpha_1, \alpha_2, \dots, \alpha_n)$, such that node i of G corresponds to node α_i of G^* . Since the rows and columns of the adjacency matrices, A and B , correspond to the nodes of G and G^* , we see that by permuting the labels (or subscripts) of the nodes results in permuting the corresponding rows and columns of the adjacency matrix. Hence, if P is the permutation matrix given by (2.7) corresponding to the permutation $(\alpha_1, \alpha_2, \dots, \alpha_n)$, then the similarity transformation, PBP' , results in permuting the labels of the nodes of the corresponding graph. That is, if B is the adjacency matrix of G^* , then $R_i(PBP')$ and $C_i(PBP')$ correspond to node α_i of G^* . Therefore, if A and B are the respective adjacency matrices of G and G^* , then the graph isomorphism problem can be reformulated as an eigenvalue problem as follows:

Definition 2.11. $G \approx G^*$ if and only if there exists a permutation matrix $P = P_{\alpha_1, \alpha_2, \dots, \alpha_n}$ (Definition 2.4) such that

$$(2.13) \quad A = PBP'.$$

Moreover, since A and B are similar, we have, from Theorem 2.3, the necessary condition that the eigenvalues of A and B must be identical.

CHAPTER III

ANALYSIS OF A NUMERICAL ALGORITHM TO DETERMINE GRAPH ISOMORPHISM

In this chapter, we present the analysis of a numerical algorithm to determine graph isomorphism for the special case where the adjacency matrices have distinct eigenvalues and the general case where the adjacency matrices may have one or more multiple eigenvalues.

3.1 Analysis of the Algorithm for Distinct Eigenvalues

The algorithm described in this section was developed by R.S. Julius and S. Charmonman [14]. Their analysis is presented here since the extension of the algorithm to include multiple eigenvalues is a generalization of their algorithm for distinct eigenvalues.

3.1.1 Preliminary Analysis Let A and B be the adjacency matrices of the simple n -node graphs G and G^* , respectively. If $G \simeq G^*$, then from (2.13) A and B are similar. Therefore, if Λ is the diagonal matrix of distinct eigenvalues and U and V are the matrices of eigenvectors, normalized to unit length, of A and B , then Theorem 2.3 gives

$$(3.1) \quad AU = UA$$

and

$$(3.2) \quad BV = VA .$$

By combining (2.13), (3.1) and (3.2), we obtain

$$(3.3) \quad PV = UD ,$$

where D is a diagonal matrix with $d_{ii} = \pm 1$, $i=1,2,\dots,n$.

Solving (3.3) for P yields

$$(3.4) \quad P = UD V^{-1} ,$$

which gives us n^2 equations relating the elements of P to those of D . By Theorem 2.1, (3.4) can be rewritten as

$$(3.5) \quad R = [U \otimes (V^{-1})'] f$$

where

$$(3.6) \quad R = R'(P)$$

and

$$(3.7) \quad f = R'(D) .$$

Since the vector f consists of the transposed rows of the diagonal matrix D , it has non-zero elements only in positions $(j-1)n + j$, $j=1,2,\dots,n$, thus only these n columns of $U \otimes (V^{-1})'$ need be considered. Therefore, we can rewrite (3.5) as

$$(3.8) \quad R = Kd ,$$

where the n^2 -by- n matrix K is obtained from $U \otimes (V^{-1})'$ by deleting all but the $(j-1)n + j$, $j=1,2,\dots,n$, columns, and the elements of d are $d_j = (D)_{jj}$, $j=1,2,\dots,n$.

If we partition K into n submatrices, K_i , $i=1,2,\dots,n$, of order n , we can write (3.8) as n systems of equations as follows:

$$(3.9) \quad R'_i(P) = K_i d , \quad i=1,2,\dots,n .$$

The matrices K_i are given by

$$(3.10) \quad K_i = [u_{i1}R'_1(V^{-1}), u_{i2}R'_2(V^{-1}), \dots, u_{in}R'_n(V^{-1})] ,$$

for $i=1,2,\dots,n$.

Using (3.9), we can determine the n rows of P and the method we use depends on whether or not all of the K_i

are singular. From (3.10), it is easily seen that since K_i is composed of multiples of the transposed rows of the non-singular matrix V^{-1} , K_i is singular if and only if $R_i(U)$ has one or more zero elements.

3.1.2 Non-Singular Procedure If we assume that at least one row of U , $R_s(U)$, has all non-zero elements then the s^{th} set of equations in (3.9) can be solved explicitly for d giving

$$(3.11) \quad d = K_s^{-1} R'_s(P) .$$

The non-singular procedure is obtained by substituting (3.11) in (3.9), which gives

$$(3.12) \quad R'_i(P) = K_i K_s^{-1} R'_s(P) , \quad i=1,2,\dots,n .$$

Since P is a permutation matrix, $R_s(P)$ must be one of the unit row vectors e_j , $j=1,2,\dots,n$, where $(e_j)_i = \delta_{ji}$. If we let P_t be the matrix whose s^{th} row is e_t , then (3.12) becomes

$$(3.13) \quad R'_i(P_t) = C_t(K_i K_s^{-1}) , \quad i=1,2,\dots,n .$$

Combining equations (3.10) and (3.13) results in

$$(3.14) \quad (P_t)_{ij} = \sum_{k=1}^n \frac{u_{ik}}{u_{sk}} v_{tk} (V^{-1})_{kj} \quad , \quad i, j, t=1, 2, \dots, n \quad .$$

The n matrices generated by (3.14) must be tested to determine whether or not they are permutation matrices.

3.1.3 Singular Procedure If all of the K_i are singular, then there is at least one zero column in each K_i . If, for $i=s$, $C_j(K_s) = 0$, then $R_s(P)$ does not depend on d_j . Thus, in the s^{th} set of equations (3.9) we can set d_j to zero without destroying the equality. Subsequently, we will interpret $d_j = 0$ to mean d_j is not defined.

If we define the n matrices L_i of order n to be

$$(3.15) \quad L_i = \begin{bmatrix} u_{i1}^+ & c_1'(V) \\ u_{i2}^+ & c_2'(V) \\ & \vdots \\ u_{in}^+ & c_n'(V) \end{bmatrix} \quad , \quad i=1, 2, \dots, n \quad ,$$

then from (3.10) and (3.15) it can be easily verified that

$L_i K_i$, $i=1,2,\dots,n$, differs from I in that $(L_i K_i)_{jj} = 0$ if and only if $C_j(K_i) = 0$. Similarly, $(L_i K_i d)_j = d_j$ when $C_j(K_i) \neq 0$, and $(L_i K_i d)_j = 0$ when $C_j(K_i) = 0$. Consequently, multiplying (3.9) on both sides by L_i gives

$$(3.16) \quad (L_i R_i'(P))_j = d_j \quad \text{or} \quad 0, \quad i,j=1,2,\dots,n.$$

However, since $R_i(P) = e_j$ for some j , we have

$$(3.17) \quad L_i R_i'(P) = C_j(L_i)$$

and if $R_i(P) = e_s$ and $R_j(P) = e_t$ then from (3.16) and (3.17) we must have that $C_s(L_i) \sim C_t(L_j)$. Moreover, from (3.16) the elements of $C_s(L_i)$ and $C_t(L_j)$ must be ± 1 or 0 .

Thus the singular procedure consists of generating the n matrices L_i from (3.15) and then searching for n columns, one from each L_i , that satisfy the following conditions:

$$1) \quad C_{\alpha_1}(L_1) \sim C_{\alpha_2}(L_2) \sim \dots \sim C_{\alpha_n}(L_n)$$

$$2) \quad \alpha_i \neq \alpha_j \quad \text{for } i \neq j$$

(3.18)

$$3) \quad (C_{\alpha_i}(L_i))_j = \pm 1 \quad \text{or} \quad 0, \quad i, j=1, 2, \dots, n$$

$$4) \quad \text{If } (C_{\alpha_i}(L_i))_j = 0, \text{ then } (L_i)_{jk} = 0,$$

for $k=1, 2, \dots, n$.

Every set of columns satisfying (3.18) yields a matrix given by

$$(3.19) \quad R_i(P) = e_{\alpha_i}, \quad i=1, 2, \dots, n$$

which is a permutation matrix satisfying (2.13). If no set of columns exists that satisfies (3.18), then $A \neq \text{PBP}'$ and hence $G \neq G^*$.

It should be noted here that the singular procedure is more general than the non-singular procedure and may

be used even if not all of the K_i are singular.

3.2 Analysis of the Algorithm for Multiple Eigenvalues

In this section, we develop the extension of the algorithm presented in Section 3.1 to include the case where the adjacency matrices have multiple eigenvalues. Since the algorithm for multiple eigenvalues is a generalization of the singular procedure discussed in Section 3.1, it may be used for the case where the eigenvalues are distinct.

3.2.1 Preliminary Analysis Let A and B be the adjacency matrices of the n -node graphs G and G^* . If $G \approx G^*$ and U and V are the matrices of orthonormalized eigenvectors of A and B , then (2.13), (3.1) and (3.2) are still valid. Let A have ρ distinct eigenvalues, λ_i , $i=1,2,\dots,\rho$, with multiplicity n_i , $i=1,2,\dots,\rho$, and assume that the λ_i are ordered according to increasing multiplicity, that is, $n_i \leq n_{i+1}$, $i=1,2,\dots,\rho-1$. Equations (3.3) and (3.4) still hold but since we have multiple eigenvalues, an eigenvector of A corresponding to λ_k with multiplicity n_k is no longer a permutation of one eigenvector of B but may be a permutation of a linear combination of all the eigenvectors of B that correspond to λ_k . Therefore, D is no longer strictly diagonal but is block diagonal of the form

$$(3.20) \quad D = \begin{bmatrix} \Delta_1 & & & & \\ & \Delta_2 & & & \\ & & \ddots & & \\ & & & \ddots & \\ 0 & & & & \Delta_\rho \end{bmatrix},$$

where the ρ submatrices Δ_i are of order n_i and give the explicit linear combination. Equation (3.5),

$$(3.5) \quad R = [U \otimes (V^{-1})'] f,$$

still holds where R and f are given by (3.6) and (3.7), but since D is block diagonal, the column vector f is of the form

$$(3.21) \quad f' = \left[[R_1(\Delta_1), \bar{0}], [R_2(\Delta_1), \bar{0}], \dots, [R_{n_1}(\Delta_1), \bar{0}], \right. \\ \left. [\bar{0}, R_1(\Delta_2), \bar{0}], \dots, [\bar{0}, R_{n_\rho}(\Delta_\rho)] \right].$$

Deleting the columns of $U \otimes (V^{-1})'$ corresponding to zero elements of f , we can rewrite (3.5) as

$$(3.22) \quad R = Kd$$

where d is a column vector with $\sum_{i=1}^p n_i^2$ elements given by

$$(3.23) \quad d' = [R_1(\Delta_1), R_2(\Delta_1), \dots, R_{n_1}(\Delta_1),$$

$$R_1(\Delta_2), \dots, R_{n_p}(\Delta_p)]$$

and K is of dimension n^2 -by- $\sum_{i=1}^p n_i^2$ given by

$$K = \left[[C_1(U), C_2(U), \dots, C_{n_1}(U)] \otimes [C_1((V^{-1})'), C_2((V^{-1})'), \right.$$

$$\dots, C_{n_1}((V^{-1})')], [C_{n_1+1}(U), \dots, C_{n_1+n_2}(U)]$$

$$(3.24)$$

$$\otimes [C_{n_1+1}((V^{-1})'), \dots, C_{n_1+n_2}((V^{-1})')],$$

$$\dots, [C_{n-n_p+1}(U), \dots] \otimes [\dots, C_n((V^{-1})')]\quad .$$

Partitioning K into n submatrices, we have

$$(3.25) \quad R'_i(P) = K_i d, \quad i=1,2,\dots,n,$$

where the K_i are given explicitly by

$$(3.26) \quad K_i = \begin{bmatrix} [u_{i1}, u_{i2}, \dots, u_{in_1}] \\ \otimes [c_1((V^{-1})'), \dots, c_{n_1}((V^{-1})')], \\ \dots, [u_{i, n-n_\rho+1}, \dots] \otimes [\dots, c_n((V^{-1})')] \end{bmatrix},$$

for $i=1,2,\dots,n$.

If we now define the n matrices L_i to be

$$(3.27) \quad L_i = \left[\begin{array}{cc} \begin{pmatrix} u_{i1}^+ \\ u_{i2}^+ \\ \vdots \\ u_{in_1}^+ \end{pmatrix} & \otimes \begin{pmatrix} C_1'(V) \\ C_2'(V) \\ \vdots \\ C_{n_1}'(V) \end{pmatrix} \\ \vdots & \vdots \\ \begin{pmatrix} u_{i,n-n_\rho+1}^+ \\ \vdots \\ u_{in}^+ \end{pmatrix} & \otimes \begin{pmatrix} C_{n-n_\rho+1}'(V) \\ \vdots \\ C_n'(V) \end{pmatrix} \end{array} \right], \quad i=1,2,\dots,n,$$

we can multiply both sides of (3.25) by L_i to obtain the equation

$$(3.28) \quad L_i R_i'(P) = L_i K_i d, \quad i=1,2,\dots,n.$$

Multiplying (3.26) by (3.27) yields

$$(3.29) \quad L_i K_i = \begin{bmatrix} M_{i1} & & & 0 \\ & M_{i2} & & \\ & & \ddots & \\ 0 & & & M_{i\rho} \end{bmatrix}, \quad i=1,2,\dots,n,$$

which is block diagonal and the submatrices M_{ik} are of order n_k^2 . Furthermore, M_{ik} , $k=1,2,\dots,\rho$, may be written as

$$(3.30) \quad M_{ik} = m_{ik} \otimes I_{n_k},$$

for $k=1,2,\dots,\rho$; $i=1,2,\dots,n$, where m_{ik} is given by

$$(3.31) \quad m_{ik} = \begin{bmatrix} \begin{pmatrix} u_{i,t+1}^+ \\ u_{i,t+2}^+ \\ \vdots \\ u_{i,t+n_k}^+ \end{pmatrix} \otimes (u_{i,t+1}, u_{i,t+2}, \dots, u_{i,t+n_k}) \end{bmatrix},$$

$$k=1,2,\dots,\rho, \quad i=1,2,\dots,n,$$

and

$$(3.32) \quad t = \sum_{j=1}^{k-1} n_j .$$

If d is written in the form

$$(3.33) \quad d = \begin{bmatrix} R'(\Delta_1) \\ R'(\Delta_2) \\ \vdots \\ R'(\Delta_\rho) \end{bmatrix} ,$$

then

$$(3.34) \quad L_i K_i d = \begin{bmatrix} M_{i1} R'(\Delta_1) \\ M_{i2} R'(\Delta_2) \\ \vdots \\ M_{i\rho} R'(\Delta_\rho) \end{bmatrix} , \quad i=1,2,\dots,n .$$

Now, from (3.30), (3.31) and (3.34), for a fixed k we have

$$(3.35) \quad M_{1k} R'(\Delta_k) = \begin{bmatrix} R_1(m_{1k}) C_1(\Delta_k) \\ R_1(m_{1k}) C_2(\Delta_k) \\ \vdots \\ R_1(m_{1k}) C_{n_k}(\Delta_k) \\ R_2(m_{1k}) C_1(\Delta_k) \\ \vdots \\ R_{n_k}(m_{1k}) C_{n_k}(\Delta_k) \end{bmatrix}$$

and from (3.27) and (3.28),

$$(3.36) \quad M_{1k} R'(\Delta_k) = \left[\begin{pmatrix} u_{1,t+1}^+ \\ u_{1,t+2}^+ \\ \vdots \\ u_{1,t+n_k}^+ \end{pmatrix} \otimes \begin{pmatrix} C'_{t+1}(V) \\ C'_{t+2}(V) \\ \vdots \\ C'_{t+n_k}(V) \end{pmatrix} \right] R'_1(P) .$$

Combining (3.35) and (3.36), the equations for $C_j(\Delta_k)$, $j=1,2,\dots,n_k$, are

$$(3.37) \quad u_{i,t+s}^+ C_{t+j}'(V) R_i'(P) = R_s(m_{ik}) C_j(\Delta_k) , \quad s=1,2,\dots,n_k ,$$

where t is given by (3.32). For a given j , (3.37) gives n_k equations for $C_j(\Delta_k)$ which differ from each other by a constant multiplier, which may be zero.

Therefore, we need to choose a non-null set of equations for $j=1,2,\dots,n_k$ in order to solve for Δ_k .

If we define an array Q of dimension n -by- ρ such that

$$(3.38) \quad q_{ik} = \begin{cases} t+r , & \text{if } u_{i,t+j}=0, j=1,2,\dots,r-1, u_{i,t+r} \neq 0 \\ t+n_k , & \text{if } u_{i,t+j}=0, j=1,2,\dots,n_k \end{cases} ,$$

$$k=1,2,\dots,\rho , \quad i=1,2,\dots,n ,$$

where t is defined by (3.32), then by deleting all rows of L_i except those corresponding to $u_{i,q_{ik}}$, $k=1,2,\dots,\rho$, we obtain

$$(3.39) \quad \bar{L}_i = \begin{bmatrix} u_{i,q_{i1}}^+ & C_1'(V) \\ u_{i,q_{i1}}^+ & C_2'(V) \\ \vdots & \vdots \\ u_{i,q_{i1}}^+ & C_{n_1}'(V) \\ \vdots & \vdots \\ u_{i,q_{i\rho}}^+ & C_n'(V) \end{bmatrix}, \quad i=1,2,\dots,n,$$

which is of order n . Then (3.28) reduces to

$$(3.40) \quad \bar{L}_i R_i'(P) = \bar{L}_i K_i d = \begin{bmatrix} R_{(q_{i1})-t_1}^{(m_{i1})} C_1(\Delta_1) \\ \vdots \\ R_{(q_{i1})-t_1}^{(m_{i1})} C_{n_1}(\Delta_1) \\ R_{(q_{i2})-t_2}^{(m_{i2})} C_1(\Delta_2) \\ \vdots \\ R_{(q_{i\rho})-t_\rho}^{(m_{i\rho})} C_{n_\rho}(\Delta_\rho) \end{bmatrix},$$

for $i=1,2,\dots,n$,

where

$$(3.41) \quad t_1 = 0, \quad t_k = \sum_{i=1}^{k-1} n_i, \quad k=2,3,\dots,\rho.$$

Therefore, for a given k , by choosing a set of n_k i 's, $\gamma_1, \gamma_2, \dots, \gamma_{n_k}$, such that $R_{(q_{\gamma_j, k}) - t_k}^{(m_{\gamma_j, k})}$, $j=1,2,\dots,n_k$, are linearly independent, we can solve (3.40) for Δ_k by choosing a set of unit row vectors for n_k rows of P . For example, if $k=1$, we obtain the equations

$$(3.42)$$

$$R_{(q_{\gamma_1, 1}) - t_1}^{(m_{\gamma_1, 1})} C_j(\Delta_1) = (\bar{L}_{\gamma_1} R'_{\gamma_1}(P))_j$$

$$R_{(q_{\gamma_2, 1}) - t_1}^{(m_{\gamma_2, 1})} C_j(\Delta_1) = (\bar{L}_{\gamma_2} R'_{\gamma_2}(P))_j$$

$$, \quad j=1,2,\dots,n_1.$$

.

$$R_{(q_{\gamma_{n_1}, 1}) - t_1}^{(m_{\gamma_{n_1}, 1})} C_j(\Delta_1) = (\bar{L}_{\gamma_{n_1}} R'_{\gamma_{n_1}}(P))_j$$

Since the matrix of coefficients on the left-hand side of (3.42) is independent of j , we can write (3.42) as a

matrix equation as follows

$$(3.43) \quad \begin{bmatrix} R(q_{\gamma_1,1}) - t_1(m_{\gamma_1,1}) \\ R(q_{\gamma_2,1}) - t_1(m_{\gamma_2,1}) \\ \vdots \\ R(q_{\gamma_{n_1},1}) - t_1(m_{\gamma_{n_1},1}) \end{bmatrix} \Delta_1 = \begin{bmatrix} R_{\gamma_1}(P) \bar{L}'_{\gamma_1,1} \\ R_{\gamma_2}(P) \bar{L}'_{\gamma_2,1} \\ \vdots \\ R_{\gamma_{n_1}}(P) \bar{L}'_{\gamma_{n_1},1} \end{bmatrix},$$

where

$$(3.44) \quad \bar{L}'_{ij} = [c_{t_j+1}(\bar{L}'_i), c_{t_j+2}(\bar{L}'_i), \dots, c_{t_j+n_j}(\bar{L}'_i)].$$

In general, we choose n_k γ_j 's such that

$R(q_{\gamma_j,k}) - t_k(m_{\gamma_j,k})$, $j=1,2,\dots,n_k$, are linearly independent

and solve the system

(3.45)

$$\Delta_k = \begin{bmatrix} R_{(\gamma_1, k)} - t_k^{(m_{\gamma_1 k})} \\ R_{(\gamma_2, k)} - t_k^{(m_{\gamma_2 k})} \\ \vdots \\ R_{(\gamma_{n_k}, k)} - t_k^{(m_{\gamma_{n_k} k})} \end{bmatrix} = \begin{bmatrix} R_{\gamma_1}^{(P)} \bar{L}'_{\gamma_1 k} \\ R_{\gamma_2}^{(P)} \bar{L}'_{\gamma_2 k} \\ \vdots \\ R_{\gamma_{n_k}}^{(P)} \bar{L}'_{\gamma_{n_k} k} \end{bmatrix}, \quad k=1, 2, \dots, \rho,$$

by choosing unit row vectors for $R_{\gamma_i}^{(P)}$, $i=1, 2, \dots, n_k$.

If we define the matrix \bar{M} of order n to be

and consider \bar{M} to be partitioned into ρ submatrices, \bar{M}_j , of dimension n -by- n_j where

(3.47)

$$\bar{M}_j = [C_{t_j+1}(\bar{M}), C_{t_j+2}(\bar{M}), \dots, C_{t_j+n_j}(\bar{M})] , \quad j=1,2,\dots,\rho ,$$

and t_j is given by (3.41), then from (3.31), (3.38) and (3.46) we have

$$(3.48) \quad R_{(q_{\gamma_j k})-t_k(m_{\gamma_j k})} = R_{\gamma_j}(\bar{M}_k) .$$

Hence, (3.45) may be rewritten as

$$(3.49) \quad \begin{bmatrix} R_{\gamma_1}(\bar{M}_k) \\ R_{\gamma_2}(\bar{M}_k) \\ \vdots \\ R_{\gamma_{n_k}}(\bar{M}_k) \end{bmatrix} \Delta_k = \begin{bmatrix} R_{\gamma_1}(P)\bar{L}'_{\gamma_1 k} \\ R_{\gamma_2}(P)\bar{L}'_{\gamma_2 k} \\ \vdots \\ R_{\gamma_{n_k}}(P)\bar{L}'_{\gamma_{n_k} k} \end{bmatrix} , \quad k=1,2,\dots,\rho .$$

Since the matrices of eigenvectors U and V of real symmetric matrices are orthogonal (Theorem 2.2), we have from (2.6) and (3.3) that

$$(3.50) \quad D = U'PV$$

and hence from Theorem 2.2 and equation (2.8),

$$(3.51) \quad D'D = I .$$

Combining (3.20) and (3.51) gives us a necessary condition that Δ_i must satisfy

$$(3.52) \quad \Delta_i' \Delta_i = I_{n_i} , \quad i=1,2,\dots,\rho .$$

Therefore, if n_k rows of P are chosen and $\Delta_k' \Delta_k \neq I_{n_k}$ then we must change our choice of unit vectors for these n_k rows of P . If, however, $\Delta_k' \Delta_k = I_{n_k}$, then that particular choice of n_k rows of P may yield a permutation matrix that satisfies (2.13).

If m , $n_\rho \leq m \leq n$, rows of P need be chosen in order to compute Δ_k , $k=1,2,\dots,\rho$, and if these m rows yield

$\rho \Delta_k$'s such that (3.52) is satisfied, then we ultimately obtain a permutation matrix P that satisfies (2.13).

If n_i rows of P have been chosen and Δ_k , $k=1,2,\dots,i$, have been computed satisfying (3.52), we can utilize a modification of (3.49) to determine possible choices of unit row vectors for subsequent rows of P that must be chosen. Let us assume that $R_{\gamma_j}(P)$, $j=1,2,\dots,n_i$, have been chosen and Δ_k , $k=1,2,\dots,i$, computed. Furthermore, we assume that Δ_k , $k=1,2,\dots,i$, satisfies (3.52) and that for some k , $R_{\gamma_k}(P)$, $\gamma_k \neq \gamma_j$, $j=1,2,\dots,n_i$, must be chosen in order to compute Δ_{i+1} . If there exists a permutation matrix P satisfying (2.13) that corresponds to our choice of unit vectors for $R_{\gamma_j}(P)$, $j=1,2,\dots,n_i$, then from (3.38), (3.46) and (3.49) we must have

$$(3.53) \quad [R_{\gamma_k}(\bar{M}_1), R_{\gamma_k}(\bar{M}_2), \dots, R_{\gamma_k}(\bar{M}_i)] \begin{bmatrix} \Delta_1 & & & 0 \\ & \Delta_2 & & \\ & & \ddots & \\ 0 & & & \Delta_i \end{bmatrix}$$

$$\sim [R_t(\bar{L}'_{\gamma_k 1}), R_t(\bar{L}'_{\gamma_k 2}), \dots, R_t(\bar{L}'_{\gamma_k i})]$$

for some t . If $e_t \neq R_{\gamma_j}(P)$, $j=1,2,\dots,n_i$, for the set

of t 's which satisfy (3.53), then the only possible choices of unit row vectors for $R_{\gamma_k}(P)$, based on our previous assumptions, are

$$(3.54) \quad R_{\gamma_k}(P) = e_t .$$

Thus, we can use (3.53) to determine future choices for $R_1(P)$ and hence, reduce the degree of arbitrariness in the algorithm. If however, (3.53) does not hold for some t , then no permutation matrix exists corresponding to our choice of unit row vectors for $R_{\gamma_j}(P)$, $j=1,2,\dots,n_1$.

3.2.2 Procedure for Multiple Eigenvalues The numerical algorithm for multiple eigenvalues consists firstly of generating the matrix Q according to (3.38). Secondly, we generate the n matrices, \bar{L}_i , of order n , from

$$(3.55) \quad \bar{L}_i = \left[u_{iq_{i1}}^+ [C_1(v), \dots, C_{n_1}(v)], u_{iq_{i2}}^+ [C_{n_1+1}(v), \dots], \right. \\ \left. \dots, u_{iq_{ip}}^+ [\dots, C_n(v)] \right], \quad i=1,2,\dots,n ,$$

and \bar{M} from (3.46).

From \bar{M}_j , we choose n_j linearly independent rows, $R_{\gamma_i}(\bar{M}_j)$, $i=t_j+1, t_j+2, \dots, t_j+n_j$, for $j=1, 2, \dots, \rho$ and call these ρ submatrices, which are of order n_j , T_j , $j=1, 2, \dots, \rho$. Since T_j is precisely the coefficient matrix of Δ_j in (3.49); we compute T_j^{-1} , $j=1, 2, \dots, \rho$, in order to solve for Δ_j .

The procedure then consists of selecting a set of n_1 unit row vectors for $R_{\gamma_i}(P)$, $i=1, 2, \dots, n_1$, and computing Δ_1 from (3.49) until (3.52) is satisfied. Then, if necessary, determine possible choices for $R_{\gamma_i}(P)$, $i=t_j+1, t_j+2, \dots, t_j+n_j$, using (3.53) and (3.54) and compute Δ_j , $j=2, 3, \dots, \rho$. For each j , if (3.52) holds, then we proceed to compute Δ_{j+1} in the same manner. If, however, (3.52) is not satisfied, or (3.52) is satisfied and (3.53) and (3.54) do not hold, then we must change our selection of unit vectors for $R_{\gamma_i}(P)$, $i=t_j+1, t_j+2, \dots, t_j+n_j$, and re-calculate all Δ_i , $1 \leq i \leq j$, which were computed from the previous selection.

When all Δ_i , $i=1, 2, \dots, \rho$, have been computed satisfying (3.52) for a given set of unit vectors for m rows of P , we generate all permutation matrices based on this selection using (3.53) and (3.54). The algorithm then consists of backtracking and changing our selection of unit row vectors until all permutation matrices are

ultimately generated satisfying (2.13).

If at least one P is generated by the algorithm, then $G \approx G^*$ and if no P is generated, then $G \not\approx G^*$. Moreover, if no set of unit row vectors, e_t , for m rows of P exists such that (3.52) holds for $i=1,2,\dots,\rho$, then $G \not\approx G^*$. Hence, the algorithm for multiple eigenvalues halts in one of two states indicating whether or not $G \approx G^*$.

CHAPTER IV

NUMERICAL EXPERIMENTS

The numerical algorithm for determining graph isomorphisms described in Section 3.2 of Chapter III was programmed in Fortran IV source language (see Appendix) and tested successfully on the IBM 360/67 computer installation at the University of Alberta. In an attempt to determine the efficiency of the algorithm, several types of graphs were used as test data and the program was timed using an assembly language subroutine CS019A, provided by the Computing Centre, which accesses the OS/360 interval timer.

4.1 Test Data

The test data consist of:

- i) Isomorphic graphs with distinct eigenvalues;
- ii) Isomorphic graphs with multiple eigenvalues;
- iii) Non-isomorphic graphs with different eigenvalues; and,
- iv) Non-isomorphic graphs with identical eigenvalues.

Several graphs of these four types are presented in Tables 4.1, 4.2, 4.3 and 4.4. Their corresponding adjacency matrices are not given as they can be obtained from the graphs by using Definition 2.8. For example, the complete graph with 3 nodes and its corresponding adjacency matrix

is given in Figure 4.1.

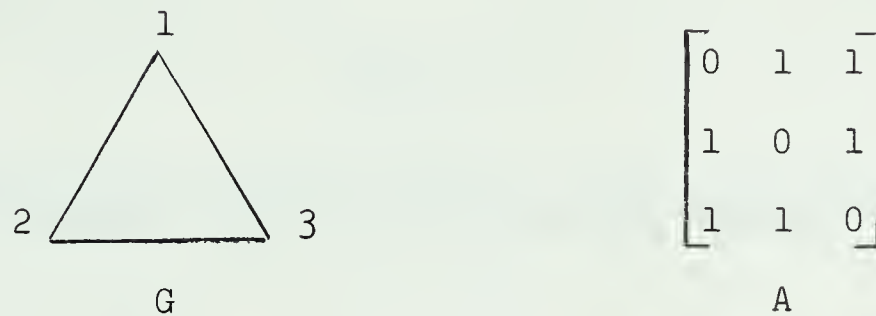


Figure 4.1 The Complete 3-Node Graph



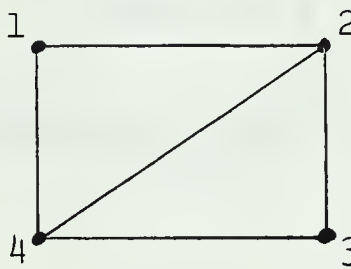
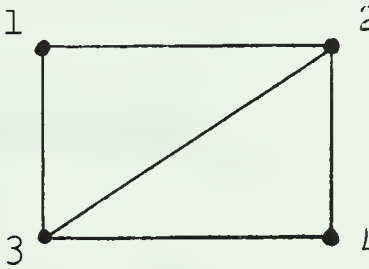
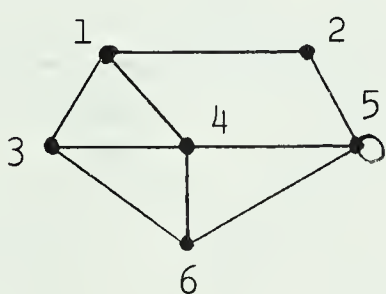
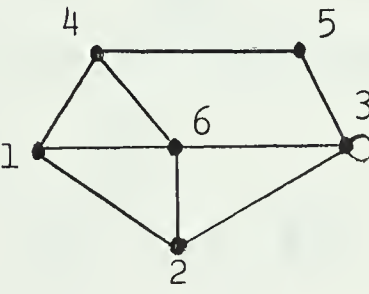
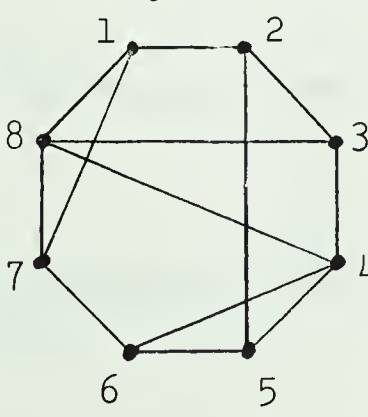
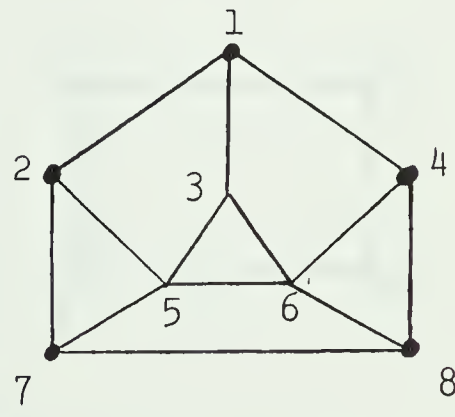
4.1.1 Isomorphic Graphs with Distinct Eigenvalues

Since the procedure for multiple eigenvalues is a generalization of the singular procedure for distinct eigenvalues, the algorithm for multiple eigenvalues may be used to determine isomorphisms between graphs with distinct eigenvalues. Consequently, four graphs, G_1 , G_2 , G_3 and G_4 , with distinct eigenvalues are used as test data. G_1^* , G_2^* and G_3^* are differently labelled versions of G_1 , G_2 and G_3 . G_1 and G_2 are taken from [14], G_3 is a modified undirected graph corresponding to the directed graph, G , in Figure 2.1 in [24] and G_4 and G_4^* are modified undirected graphs corresponding to the directed graphs, G_1 and G_2 , in Figure 3.2 in [24]. The modified undirected graphs are

obtained by deleting all parallel edges from the associated undirected graphs of the corresponding directed graphs.

Table 4.1

Isomorphic Graphs with Distinct Eigenvalues

i	n	G_i	G_i^*
1	3		
2	4		
3	6		
4	8		

4.1.2 Isomorphic Graphs with Multiple Eigenvalues

The graphs G_i , $i=5,6,\dots,19$ have multiple eigenvalues and the eigenvalues of G_i^* are identical to those of G_i . G_5^* , G_6^* and G_8^* are differently labelled versions of G_5 , G_6 and G_8 , while G_7^* and G_9^* are identical to G_7 and G_9 respectively. G_9 is the associated undirected graph of the digraph given in Figure 5 in [27]. The graphs G_i and G_i^* , $i=10,11,\dots,19$, are the complete graphs with 3,4,5,6,7,10,15,20,25 and 30 nodes respectively.

Table 4.2

Isomorphic Graphs with Multiple Eigenvalues

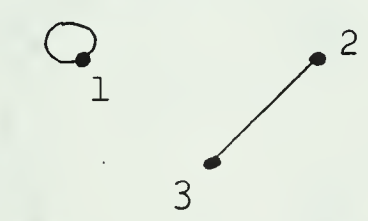
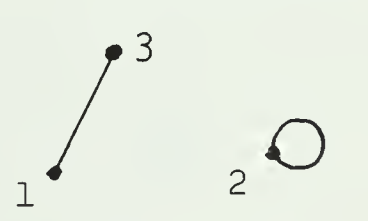
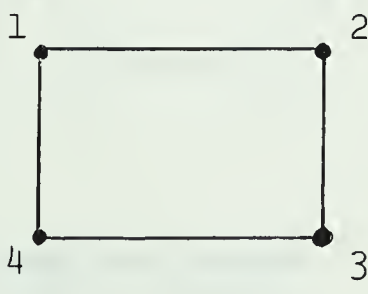
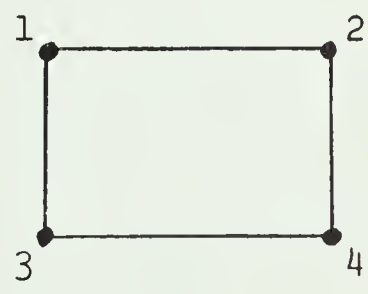
i	n	G_i	G_i^*
5	3		
6	4		

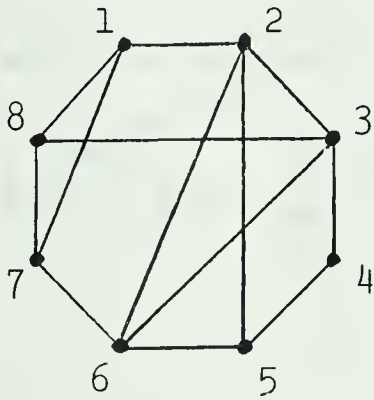
Table 4.2 (Continued)

i	n	G_i	G_i^*
7	6		G_6
8	7		
9	10		G_9
10-19	3, 4, 5, 6, 7, 10, 15, 20, 25, 30	Complete Graph with n nodes	G_i

4.1.3 Non-Isomorphic Graphs with Different Eigenvalues G_{20} is the modified undirected graph corresponding to G_3 of Figure 3.2 in [24]. G_{21} is G_{20} , G_{20}^* is G_4 and G_{21}^* is G_4^* .

Table 4.3

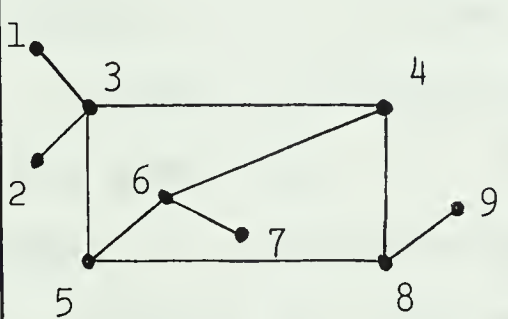
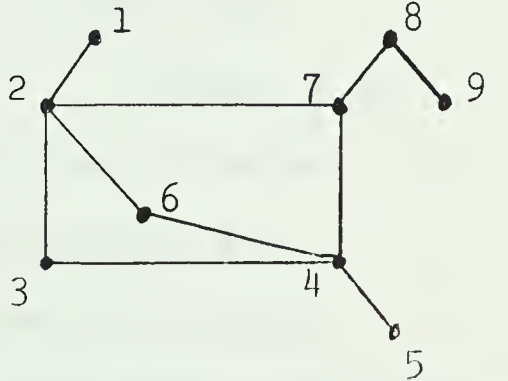
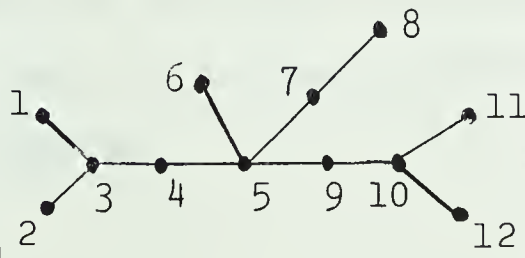
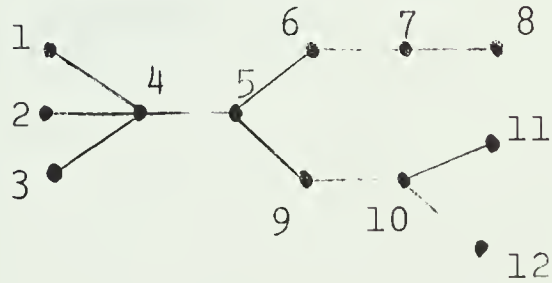
Non-Isomorphic Graphs with Different Eigenvalues

i	n	G_i	G_i^*
20	8		G_4
21	8	G_{20}	G_4^*

4.1.4 Non-Isomorphic Graphs with Identical Eigenvalues G_{22} , G_{22}^* , G_{23} and G_{23}^* were discovered by a computer search technique and are taken from a paper by Turner [26]. The eigenvalues of G_i are identical to those of G_i^* but $G_i \neq G_i^*$, $i=22,23$.

Table 4.4

Non-Isomorphic Graphs with Identical Eigenvalues

i	n	G_i	G_i^*
22	9		
23	12		

4.2 Summary of the Results

The adjacency matrices A_i and B_i , $i=1,2,\dots,23$, corresponding to the graphs G_i and G_i^* , $i=1,2,\dots,23$, are constructed and used as input to the program. For each pair of graphs, G_i and G_i^* , $i=1,2,\dots,23$, the time required by the program to determine whether or not $G_i \approx G_i^*$, $i=1,2,\dots,23$, is obtained using the timer routine. The results of the timing are given in Tables 4.5, 4.6 and 4.7. Table 4.5 gives the time required by the program to determine that $G_i \approx G_i^*$ for $i=1,2,\dots,19$. Table 4.6 gives the time required to determine that $G_i \not\approx G_i^*$ for $i=20,21,22,23$. Table 4.7 gives the total time required to generate all permutation matrices satisfying (2.13) for graphs G_i and G_i^* , $i=1,2,\dots,14$. In the following tables, i stands for the number of the graph pair, n the number of nodes and Φ the number of permutation matrices that are generated. T_1 includes the time required to compute the eigenvalues and eigenvectors of the adjacency matrices while T_2 does not include this time. Both T_1 and T_2 are average times, in seconds, for five separate test runs on the computer.

Table 4.5

Time Required to Determine that $G_1 \approx G_1^*$

i	n	Φ	T_1 (secs)	T_2 (secs)
1	3	1	0.0219	0.0088
2	4	1	0.0586	0.0104
3	6	1	0.1661	0.0219
4	8	1	0.3341	0.0339
5	3	1	0.0306	0.0077
6	4	1	0.0508	0.0113
7	6	1	0.1257	0.0212
8	7	1	0.2246	0.0281
9	10	1	0.5605	0.0560
10	3	1	0.0240	0.0075
11	4	1	0.0344	0.0109
12	5	1	0.0545	0.0161
13	6	1	0.0767	0.0230
14	7	1	0.1146	0.0323
15	10	1	0.3012	0.0765
16	15	1	0.9875	0.2236
17	20	1	2.2035	0.5126
18	25	1	3.9710	0.9524
19	30	1	8.2890	1.6174

Table 4.6

Time Required to Determine that $G_i \neq G_i^*$

i	n	Φ	$T_1(\text{secs})$	$T_2(\text{secs})$
20	8	0	0.2949	0.0023
21	8	0	0.2981	0.0023
22	9	0	0.4214	0.0331
23	12	0	0.9176	0.0645

Table 4.7

Time Required to Generate all Permutation

Matrices P such that $A_i = PB_iP'$

i	n	Φ	$T_1(\text{secs})$	$T_2(\text{secs})$
1	3	2	0.0283	0.0131
2	4	4	0.0714	0.0217
3	6	1	0.1711	0.0251
4	8	2	0.3603	0.0583
5	3	2	0.0352	0.0107
6	4	8	0.0772	0.0363
7	6	48	0.4830	0.3759
8	7	14	0.4196	0.2215
9	10	20	1.1750	0.6680
10	3	6	0.0352	0.0173
11	4	24	0.1010	0.0758
12	5	120	0.5737	0.5329
13	6	720	4.7717	4.7226
14	7	5040	48.8116	48.7396

CHAPTER V

CONCLUSIONS AND SUGGESTIONS FOR FUTURE RESEARCH

5.1 Conclusions

The following conclusions are based on the results of the numerical experiments:

1. The numerical algorithm for determining graph isomorphisms developed in Section 3.2 of Chapter III is capable of determining isomorphisms between graphs with distinct eigenvalues or between graphs with one or more multiple eigenvalues. Moreover, the algorithm can detect non-isomorphic graphs with identical eigenvalues as well as non-isomorphic graphs with different eigenvalues.
2. The total time required by the algorithm to determine whether or not two graphs are isomorphic is comprised, primarily, of the time used in computing the eigenvalues and eigenvectors of the adjacency matrices. For the test data, this time is 60-99% of the total time required. Consequently, since the eigenvalues and eigenvectors are obtained by Jacobi's method, the total time required by the algorithm to determine whether or not two graphs are isomorphic can be significantly reduced by implementing a faster routine, such as Householder's method, in order to compute the eigenvalues and eigenvectors.

3. The time required by the algorithm to generate all permutation matrices for a given pair of isomorphic graphs depends on the number of nodes, n , and on the degree of symmetry of the graphs. However, since the complete n -node graph has $n!$ permutation matrices, the upper bound on the time required to generate all permutation matrices for a pair of n -node graphs grows as $n!$. Thus it is virtually impossible to develop an efficient algorithm to generate all permutation matrices for a given pair of isomorphic graphs.
4. The algorithm requires up to $n^3 + \frac{13}{2}n^2 + \frac{13}{2}n$ words of storage for data to determine whether or not two graphs are isomorphic. Therefore, for large n a considerable amount of storage space is required which may necessitate the use of peripheral storage devices.
5. Sussenguth's structure-matching algorithm is more general since it is capable of detecting sub-structure matches. Moreover, Sussenguth's algorithm handles directed graphs, in general, without modification although a transformation from directed to undirected graphs is sometimes used to aid in determining isomorphisms between directed graphs. The algorithm presented in this paper can be modified to determine complete isomorphisms for directed graphs by considering the problem

$$A'A = P(B'B)P'$$

and determining which P 's , if any, satisfy

$$A = PBP' .$$

5.2 Suggestions for Future Research

1. Since the various algorithms for determining graph isomorphisms have been programmed in different languages and tested on different computers, an accurate comparison of the efficiency of the different algorithms with respect to time has not yet been made. Moreover, the results of the numerical experiments indicate that a more thorough comparison is warranted particularly between Sussenguth's structure-matching algorithm and the numerical algorithm presented in this paper. Thus, it is suggested that the following three problems be investigated in more detail:
 - (a) Which algorithm is the most efficient algorithm for determining whether or not two graphs are isomorphic?
 - (b) Which algorithm is more efficient when all isomorphic mappings of one graph onto the other are required?
 - (c) Which algorithm is best suited for implementation in an information retrieval system taking into consideration the storage requirements for the

file structures; the time required to generate the required information to compare a query structure with those on file; and, the type of match required by the request.

2. Since the algorithm presented in this paper depends only on the eigenvalues and eigenvectors of the adjacency matrices and since the time required to compute the eigenvalues and eigenvectors represents a large proportion of the total time required by the algorithm, it is suggested that more research should be carried out in this area with particular emphasis on obtaining the eigenvalues and eigenvectors of graphs and their adjacency matrices. For example, it might be possible to obtain an explicit formula for the eigenvalues of $(0,1)$ matrices. Also, it might be possible to simplify the algorithm considerably by making a judicious choice of eigenvectors of the adjacency matrices, A and B , corresponding to an eigenvalue, λ_k , with multiplicity n_k . Furthermore, since the algorithm for distinct eigenvalues is considerably less complicated than that for multiple eigenvalues, an investigation into the graph-theoretic implications of graphs with distinct eigenvalues as opposed to graphs with multiple eigenvalues should be made. Perhaps, then, it might be possible to determine, 'a priori', whether or not a

graph has distinct or multiple eigenvalues by examining the graph itself.

3. One of the inherent problems of performing numerical computations on a digital computer is that of determining whether or not two real numbers are the same. Since this problem is critical to the numerical graph isomorphism algorithm, an investigation into the effects of zero criteria on the performance of the algorithm should be carried out. For example, the zero criterion used for the test data in this experiment was initially chosen to be the maximum absolute residual ($\max |A - UAU'|$) obtained from re-composing a matrix from its eigenvalues and eigenvectors. However, using this criterion throughout the algorithm resulted in some permutation matrices not being generated for some graphs. For the test data used, a multiplicative scale factor of five was found to be sufficient to allow all permutation matrices to be generated. Therefore, the effects of different zero criteria on the algorithm should be investigated more extensively.

BIBLIOGRAPHY

1. Ballard, D.L. and F. Neeland, "A Computer Technique for the Retrieval of Related Chemical Structures Utilizing a Special Topological Cipher", J. Chem. Doc., 3 (1963), pp. 196-201.
2. Bellman, R., Introduction to Matrix Analysis, McGraw-Hill Book Co., Inc., New York, 1960.
3. Berge, C., The Theory of Graphs, John Wiley and Sons, Inc., New York, 1962.
4. Busacker, R.G. and T.L. Saaty, Finite Graphs and Networks, McGraw-Hill Book Co., Inc., New York, 1965.
5. Cossum, W.E., G.M. Dyson, M.F. Lynch and R.N. Wolfe, "Mechanical Searching of Chemical Substructures by Means of Atom-by-Atom Matching at CAS", Mimeographed Report, Chemical Abstracts Service (1963).
6. DeBoer, J. and G.E. Uhlenbeck, Studies in Statistical Mechanics, Vol. 1, North-Holland Publishing Co., Amsterdam, 1962.
7. Dyson, G.M., W.E. Cossum, M.F. Lynch and H.L. Morgan, "Mechanical Manipulation of Chemical Structure", Inf. Storage and Retr., 1 (1963), pp. 69-99.

8. Fröberg, C.E., Introduction to Numerical Analysis, Addison-Wesley Publishing Co., Inc., Reading, Mass., 1965.
9. Frome, J., "Generic Mechanized Search System", J. Chem. Doc., 2 (1962), pp. 15-18.
10. Frome, J., "Searching Chemical Structures", J. Chem. Doc., 4 (1964), pp. 43-45.
11. Grossman, I. and W. Magnus, Groups and Their Graphs, Random House, Inc., New York, 1964.
12. Harary, F., "The Determinant of the Adjacency Matrix of a Graph", SIAM Review, 4 (1962), pp. 202-210.
13. Harary, F., R.Z. Norman and D. Cartwright, Structural Models, John Wiley and Sons, Inc., New York, 1965.
14. Julius, R.S. and S. Charmonman, "Graph Isomorphism", University of Alberta Computing Review, 1 (1967), pp. 31-45.
15. Littlewood, D., The Theory of Group Characters, Oxford University Press, London, 1958.
16. Marcus, M. and H. Minc, "Inequalities for General Matrix Functions", Bull. Amer. Math. Soc., 70 (1964), pp. 308-313.

17. Murdoch, D.C., Linear Algebra for Undergraduates, John Wiley and Sons, Inc., New York, 1957.
18. Ore, O., Graphs and Their Uses, Random House, Inc., New York, 1963.
19. Ore, O., Theory of Graphs, Amer. Math. Soc. Colloquium Publication, XXXVIII, Providence, R.I., 1962.
20. Ralston, A., A First Course in Numerical Analysis, McGraw-Hill Book Co., Inc., New York, 1965.
21. Ray, L.C. and R.A. Kirsch, "Finding Chemical Records by Digital Computers", Science, 126 (1957), pp. 814-819.
22. Salton, G. and E.H. Sussenguth, Jr., "A New Efficient Structure-Matching Procedure and Its Application to Automatic Retrieval Systems", American Documentation Institute, 26th Annual Meeting, October 1963, Chicago, Ill., pp. 143-146.
23. Salton, G. and E.H. Sussenguth, Jr., "Some Flexible Information Retrieval Systems Using Structure Matching Procedures", Proc. Spring Joint Computer Conf., Washington, D.C., April 1964, Spartan, Baltimore, Md., pp. 587-597.

24. Sussenguth, Jr., E.H., Structure Matching in Information Processing, Ph.D. Thesis, Harvard University, Cambridge, Mass., 1964.
25. Sussenguth, Jr., E.H., "A Graph-Theoretic Algorithm for Matching Chemical Structures", J. Chem. Doc., 5 (1965), pp. 36-43.
26. Turner, J., "Generalized Matrix Functions and the Graph Isomorphism Problem", SIAM J. Appl. Math., 16 (1968), pp. 520-526.
27. Unger, S.H., "GIT - A Heuristic Program for Testing Pairs of Directed Line Graphs for Isomorphism", Comm. ACM, 7 (1964), pp. 26-34.
28. Waldo, W.H., "Searching Two-Dimensional Structures by Computer", J. Chem. Doc., 2 (1962), pp. 1-6.
29. Wilkinson, J.H., The Algebraic Eigenvalue Problem, Oxford University Press, London, Eng., 1965.

APPENDIX

LISTINGS OF FORTRAN IV SUBROUTINES

The following pages consist of the source listings of the Fortran IV subroutines GRAPH1, GRAPH2, GRAPH3, MAXERR and INTCHG which together comprise the graph isomorphism algorithm described in Section 3.2 of Chapter III. MAXERR computes the maximum absolute error from re-composing a matrix from its matrix of eigenvectors and its eigenvalues. INTCHG orders the eigenvalues of a matrix according to increasing multiplicity and interchanges the eigenvectors correspondingly. The System /360 Scientific Subroutine EIGEN is used to compute the eigenvalues and eigenvectors of the adjacency matrices of G and G^* . MAXERR, INTCHG and EIGEN are called from GRAPH1. The source listing for EIGEN can be obtained from the System/360 Scientific Subroutine Package Programmer's Manual. The COMMON variable ZERO must be initialized to zero; the number of nodes, N , and the adjacency matrices, A and B , must be assigned values before calling GRAPH1. The adjacency matrices, A and B , of G and G^* are stored as vectors with the upper triangular portion stored columnwise in successive locations. If the eigenvalues of A and B are different, then the COMMON

variable RHO is zero upon exiting from GRAPH1; $G \neq G^*$;
and, GRAPH2 and GRAPH3 must be bypassed. Otherwise,
the graph isomorphism algorithm is implemented by
calling successively, the subroutine GRAPH1, GRAPH2
and GRAPH3.

A control program, together with numerical examples,
is also included.


```

      SUBROUTINE
1     GRAPH1(N,A,B,U,V,Q,M,L,P,PI,MM,ALPHA,BETA,GAMM,X)
C
C     THIS SUBROUTINE COMPUTES THE EIGENVALUES AND
C     EIGENVECTORS OF A & B BY JACOBI'S METHOD.  THE N
C     MATRICES L-BAR TRANSPOSE, M-BAR & Q ARE GENERATED.
C
      DIMENSION A(1),B(1),X(1),PI(1),MM(1),ALPHA(1),BETA(1),
1     GAMM(1),U(N,N),V(N,N),M(N,N),Q(N,N),P(N,N),
2     L(N,N,N)
      COMMON ZERO,RHO
      INTEGER P,PI,ALPHA,BETA,GAMM,Q,T,RHO
      REAL L,M
      N1=(N*(N+1))/2
C
C     CALL SSP ROUTINE, EIGEN, TO COMPUTE U,V & LAMBDA.
C
      K=0
      DO 2 J=1,N
      K=K+J
      A(K)=A(K)+N
      B(K)=B(K)+N
2     CONTINUE
      DO 3 J=1,N1
      X(J)=A(J)
3     CONTINUE
      CALL EIGEN (A,U,N,0)
C
C     CALL MAXERR TO COMPUTE ZERO TOLERANCE
C
      CALL MAXERR (A,X,U,ZERO,N)
      DO 4 J=1,N1
      X(J)=B(J)
4     CONTINUE
      CALL EIGEN (B,V,N,0)
      CALL MAXERR (B,X,V,ZERO,N)
C
C     CHECK TO SEE IF EIGENVALUES OF A & B ARE IDENTICAL
C     AND COMPUTE RHC, THE NUMBER OF DISTINCT EIGENVALUES,
C     AND MM(I), THEIR CORRESPONDING MULTIPLICITY.
C
      J=0
      RHC=1
      T=1
      K=1
      K1=N+1
      X(1)=A(1)
      X(K1)=B(1)
      IF(ZERO.LT.ABS((X(1)-X(K1))/X(1))) J=1
      DO 20 I=2,N

```



```

      K=K+1
      K1=K1+1
      X(I)=A(K)
      X(K1)=B(K)
      TEMP=X(I)-X(K1)
      IF(ZERO.LT.ABS(TEMP).AND.ZERO.LT.ABS(TEMP/X(I))) J=1
      TEMP=X(I)-X(I-1)
      IF(ZERO.GE.ABS(TEMP).OR.ZERO.GE.ABS(TEMP/X(I))) GO TO 19
      MM(RHO)=T
      RHC=RHC+1
      T=0
19  T=T+1
20  CONTINUE
      MM(RHO)=T
      N1=N+1
      N2=N+N
      IF(J.EQ.0) GO TO 21
      WRITE(6,120)
120  FORMAT(' ',' ** A AND B ARE NOT ISOMORPHIC ** ','-')
      RHC=0
      RETURN
21  CONTINUE
C
C  CALL INTCHG TO ORDER THE EIGENVALUES ACCORDING TO
C  INCREASING MULTIPLICITY
C
      CALL INTCHG (U,V,M,MM,ALPHA,GAMM,N,RHO)
C
C  GENERATE Q-MATRIX OF DIMENSION N-BY-RHO
C
      DO 29 I=1,N
      K2=0
      DO 27 J=1,RHC
      K1=K2+1
      K2=K2+MM(J)
      DO 23 K=K1,K2
      IF(ZERO.LT.ABS(U(I,K))) GO TO 25
23  CONTINUE
      Q(I,J)=K2
      GO TO 27
25  Q(I,J)=K
27  CONTINUE
29  CONTINUE
C
C  GENERATE THE N MATRICES L-BAR TRANSPOSE AND M-BAR
C
      DO 39 I=1,N
      K2=0
      DO 37 J1=1,RHC
      K1=K2+1

```



```
K2=K2+MM(J1)
IQ=Q(I,J1)
TEMP=U(I,IQ)
IF(ZERO.LT.ABS(TEMP)) TEMP=1/TEMP
DO 35 K=K1,K2
M(I,K)=U(I,K)*TEMP
DO 33 J=1,N
L(I,J,K)=V(J,K)*TEMP
33 CONTINUE
35 CONTINUE
37 CONTINUE
39 CONTINUE
RETURN
END
```



```

SUBROUTINE
1  GRAPH2(N,A,B,U,V,Q,M,L,P,IND,MM,ALPHA,BETA,GAMM,X)
C
C      THIS SUBROUTINE CHOOSES LINEARLY INDEPENDENT
C      ROWS OF M-BAR(I), I=1,2,...,RHO, AND COMPUTES THEIR
C      INVERSE USING GAUSSIAN ELIMINATION WITH PARTIAL
C      PIVOTING.  THE VECTORS ALPHA,BETA & GAMMA ARE GENERATED.
C
      DIMENSION A(1),B(1),X(1),IND(1),MM(1),ALPHA(1),BETA(1),
1      GAMM(1),U(N,N),V(N,N),M(N,N),Q(N,N),P(N,N),
2      L(N,N,N)
      COMMON ZERO,RHO
      INTEGER P,IND,ALPHA,BETA,GAMM,Q,T,RHO,T1
      REAL L,M
      DO 5 I=1,N
      ALPHA(I)=I
      BETA(I)=0
      GAMM(I)=0
      IND(I)=I
      DO 3 J=1,N
      U(I,J)=0
      V(I,J)=0
      IF(I.EQ.J) V(I,J)=1
      P(I,J)=2
3  CONTINUE
5  CONTINUE
C
C      CHOOSING LINEARLY INDEPENDENT ROWS OF M-BAR
C
      K2=0
      T=0
      DO 61 I=1,RHO
      K1=K2+1
      K2=K2+MM(I)
      IF(K2.NE.K1) GO TO 17
C
C      PROCEDURE FOR DISTINCT EIGENVALUES
C
      DO 10 J=1,N
      K=ALPHA(J)
      IF(ABS(1.-M(K,K1)).LT.ZERO) GO TO 12
10  CONTINUE
      WRITE(6,190)
190  FORMAT('  ERROR..NULL EIGENVECTOR ')
      GO TO 61
12  IF(J.LE.T) GO TO 15
      T=T+1
      IF(J.EQ.T) GO TO 15
      ALPHA(J)=ALPHA(T)
      ALPHA(T)=K

```



```

15  GAMM (K1)=K
    BETA(I)=T
    GO TO 61
C
C  PROCEDURE FOR MULTIPLE EIGENVALUES
C
17  T1=MM(I)
    IF(T.GT.(K2-K1)) GO TO 19
    T=K2-K1+1
19  BETA(I)=T
    K=0
    DO 21 J=K1,K2
      K=K+1
      ID=ALPHA(K)
      DO 20 J1=K1,K2
        U(J,J1)=M(ID,J1)
20  CONTINUE
      GAMM(J)=ID
21  CONTINUE
C
C  FINDING PIVCT ELEMENT FOR ELIMINATION
C
23  DO 25 II=K1,K2
      IND(II)=II
25  CONTINUE
      DO 51 II=K1,K2
        CMAX=ZERO
        DO 27 J=II,K2
          K=IND(J)
          TEMP=ABS(U(K,II))
          IF((CMAX-TEMP).GE.0.) GO TO 27
          CMAX=TEMP+ZERO
          ID=J
27  CONTINUE
      IF(ZERO.LT.CMAX) GO TO 41
C
C  DETERMINE WHICH DEPENDENT ROW IS TO BE REPLACED
C
      DO 31 J=II,K2
        K=IND(J)
        DO 29 J1=II,K2
          IF(ZERO.LT.ABS(U(K,J1))) GO TO 31
29  CONTINUE
      GO TO 33
31  CONTINUE
      GO TO 51
C
C  GENERATE ALPHA,BETA & GAMMA VECTORS
C
33  T1=T1+1

```



```

      IK=ALPHA(T1)
      GAMM(K)=IK
      IF(I.EQ.1) GO TO 35
      IF(T.GT.T1) GO TO 37
      IF(K.GT.K2+BETA(I-1)-BETA(I)) GO TO 35
      T=T+1
      BETA(I)=T
      GO TO 37
35  KT=K-K1+1
      ALPHA(T1)=ALPHA(KT)
      ALPHA(KT)=IK
37  DO 39 J1=K1,K2
      U(K,J1)=M(IK,J1)
      V(K,J1)=C
      IF(K.EQ.J1) V(K,J1)=1
39  CONTINUE
      GO TO 23
41  IF(II.EQ.K2) GO TO 51
      IF(II.EQ.ID) GO TO 43
C
C    INTERCHANGE ROW INDICATORS FOR ELIMINATION
C
      K=IND(ID)
      IND(ID)=IND(II)
      IND(II)=K
43  KK=II+1
      ID=IND(II)
C
C    GAUSSIAN ELIMINATION FOR MM(I) RIGHT-HAND SIDES
C
      DO 49 I1=KK,K2
      K=IND(I1)
      TEMP=U(K,I1)
      IF(TEMP.EQ.09) GO TO 49
      TEMP=-TEMP/U(ID,I1)
      U(K,I1)=0
      DO 45 J=KK,K2
      U(K,J)=U(K,J)+TEMP*U(ID,J)
45  CONTINUE
      DO 47 J=K1,K2
      V(K,J)=V(K,J)+TEMP*V(ID,J)
47  CONTINUE
49  CONTINUE
51  CONTINUE
C
C    BACK SUBSTITUTION TO OBTAIN THE INVERSE
C
      K1=MM(1)-1
      KK=K2-1
      IK=IND(K2)

```



```
DO 59 II=K1,K2
X(K2)=V(IK,II)/U(IK,K2)
DO 55 II=1,KI
K=K2-II
J1=IND(K)
X(K)=V(J1,II)
DO 53 J=K,KK
X(K)=X(K)-U(J1,J+1)*X(J+1)
53 CONTINUE
X(K)=X(K)/U(J1,K)
55 CONTINUE
C
C   INVERSE STORED IN V MATRIX
C
DO 57 J=K1,K2
V(J,II)=X(J)
57 CONTINUE
59 CONTINUE
61 CONTINUE
C
C   END OF ROUTINE
C
RETURN
END
```



```

      SUBROUTINE
1     GRAPH3(N,A,B,U,V,Q,M,L,P,PI,MM,ALPHA,BETA,GAMM,X)
C
C     THIS SUBROUTINE COMPUTES & TESTS DELTA(I), I=1,
C     2, ..., RHO, AND GENERATES ALL PERMUTATION MATRICES
C     SUCH THAT  $A=BPB^T$  IF GRAPHS G & G* ARE ISOMORPHIC.
C
      DIMENSION A(1),B(1),X(1),PI(1),MM(1),ALPHA(1),BETA(1),
1          GAMM(1),U(N,N),V(N,N),M(N,N),Q(N,N),P(N,N),
2          L(N,N,N)
      COMMON ZERO,RHO
      INTEGER P,PI,ALPHA,BETA,GAMM,Q,T,RHO,BTR,T1
      REAL L,M
      ZERO=5*ZERO
      NGI=0
      BTR=BETA(RHO)
      IM=MM(1)
      IMP=IM+1
C
C     GENERATE THE POSSIBLE CHOICES FOR THE FIRST MM(1)
C     ROWS OF P AND MAKE ASSUMPTIONS
C
      DO 5 I=1,N
      ID=ALPHA(I)
      IF(I.GT.IM) GO TO 2
      DO 1 J=1,N
      P(ID,J)=0
      IF(J.GE.I) P(ID,J)=1
1  CONTINUE
      PI(ID)=I
      GO TO 5
2  DO 3 J=1,IM
      P(ID,J)=-1
3  CONTINUE
5  CONTINUE
      I=1
      J=BETA(1)
      K=J
C
C     COMPUTE AND TEST DELTA(I)
C
7  K2=0
      DO 9 I1=1,I
      K1=K2+1
      K2=K2+MM(I1)
9  CONTINUE
      DO 13 I1=K1,K2
      ID=GAMM(I1)
      IP=PI(ID)
      DO 11 J1=K1,K2

```



```

      U(I1,J1)=L(ID,IP,J1)
11  CONTINUE
13  CONTINUE
      DO 19 I1=K1,K2
      DO 16 J1=K1,K2
      X(J1)=0
      DO 15 J2=K1,K2
      X(J1)=X(J1)+V(J1,J2)*U(J2,I1)
15  CONTINUE
16  CONTINUE
      DO 17 J2=K1,K2
      U(J2,I1)=X(J2)
17  CONTINUE
19  CONTINUE
      ERR=0
      DO 23 I1=K1,K2
      DO 22 J1=K1,K2
      TEMP=0
      DO 21 J2=K1,K2
      TEMP=TEMP+U(J2,I1)*U(J2,J1)
21  CONTINUE
      IF(I1.EQ.J1) TEMP=1.-TEMP
      ERR=AMAX1(ERR,ABS(TEMP))
22  CONTINUE
23  CONTINUE
      IF(ZERO.LT.ERR) GO TO 43
      I=I+1
      IF(I.LE.RHO) GO TO 25
      J=N
      GO TO 27

C
C   TEST TO SEE IF MORE ROWS OF P ARE REQUIRED TO COM-
C   PUTE THE NEXT DELTA
C
25  IF(BETA(I).EQ.BETA(I-1)) GO TO 7
      J=BETA(I)
27  K=K+1

C
C   CALCULATE POSSIBLE CHOICES FOR ROWS OF P THAT ARE
C   REQUIRED
C
      DO 41 T=K,J
      ID=ALPHA(T)
      DO 31 J1=1,K2
      X(J1)=0
      DO 29 J2=1,K2
      X(J1)=X(J1)+M(ID,J2)*U(J2,J1)
29  CONTINUE
31  CONTINUE
      DO 40 J1=1,N

```



```

      IP=P(ID,J1)
      IF(IP.EQ.-1) GO TO 40
      DO 37 J2=1,K2
      TEMP=L(ID,J1,J2)
      IF(TEMP.EQ.0.0) GO TO 37
      TEST=X(J2)-TEMP
      IF(ABS(TEMP)-1.) 33,33,35
33  IF(ZERO.LT.ABS(TEST)) GO TO 39
      GO TO 37
35  IF(ZERO.LT.ABS(TEST/TEMP)) GO TO 39
37  CONTINUE
      IF(IP.EQ.0) GO TO 40
      P(ID,J1)=1
      GO TO 40
39  IF(T.GT.BTR) GO TO 40
      P(ID,J1)=-1
40  CONTINUE
41  CONTINUE
      IK=ALPHA(K)
      GO TO 48

```

```

C
C  GENERATE ALL PERMUTATION MATRICES FOR THIS CHOICE OF
C  M ROWS OF P
C

```

```

43  IK=ALPHA(K)
      ID=PI(IK)
      DO 45 T=K,N
      IT=ALPHA(T)
      P(IT,ID)=2
45  CONTINUE
47  P(IK,ID)=0
48  DO 50 T=1,N
      IF(P(IK,T).EQ.1) GO TO 61
50  CONTINUE

```

```

C
C  START BACKTRACKING
C

```

```

      IF(K.EQ.1) GO TO 80
51  IF(K.GT.BETA(I-1)+1) GO TO 75
      DO 55 T1=K,N
      IT=ALPHA(T1)
      DO 55 T=K,N
      IK=ALPHA(T)
      ID=PI(IK)
      P(IT,ID)=2
55  CONTINUE
      K=K-1
      DO 57 T=1,RHC
      IF(K.LE.BETA(T)) GO TO 60
57  CONTINUE

```



```

60 I=T
   J=BETA(I)
   GO TO 43
61 IL=PI(IK)
   DO 63 IJ=1,N
   IF(PI(IJ).EQ.T) GO TO 64
63 CONTINUE
64 PI(IJ)=IL
   PI(IK)=T
   IF(K.EQ.N) GO TO 78
   II=K+1
   IF(II.EQ.1) GO TO 67
   DO 65 IT=II,N
   ID=ALPHA(IT)
   IF(P(ID,T).NE.1) GO TO 65
   P(ID,T)=0
65 CONTINUE
   GO TO 70
67 DO 69 IT=II,N
   ID=ALPHA(IT)
   P(ID,T)=0
   IF(IT.GT.IM) P(ID,T)=-1
69 CONTINUE
70 IF(K.EQ.J) GO TO 7
   K=K+1
   IK=ALPHA(K)
   GO TO 48
75 DO 76 T1=K,J
   IT=ALPHA(T1)
   DO 76 T=K,N
   IK=ALPHA(T)
   ID=PI(IK)
   IF(P(IT,ID).NE.0) GO TO 76
   P(IT,ID)=1
76 CONTINUE
   IK=ALPHA(K-1)
   ID=PI(IK)
   DO 77 T=K,J
   IT=ALPHA(T)
   IF(P(IT,ID).NE.0) GO TO 77
   P(IT,ID)=1
77 CONTINUE
   K=K-1
   GO TO 47

C
C   OUTPUT PERMUTATION MATRICES
C
78 NGI=NGI+1
   WRITE(6,200) NGI,(PI(II),II=1,N)
   GO TO 51

```



```
C
C   END OF ROUTINE
C
  80 IF(NGI.EQ.0) GO TO 83
    WRITE(6,201)
    RETURN
  83 WRITE(6,202)
    RETURN
200 FORMAT(' ',I15,'...',25I4)
201 FORMAT(' ', ' ** NO MORE ISOMORPHISMS ** '/'-'')
202 FORMAT(' ', ' ** A AND B ARE NOT ISOMORPHIC ** '/'-'')
    END
```



```
      SUBROUTINE MAXERR (A,B,C,ERR,N)
C
C      THIS SUBROUTINE CALCULATES THE MAXIMUM RESIDUAL
C      FROM RE-COMPOSING A MATRIX,B, FROM ITS MATRIX OF
C      EIGENVALUES,A, AND EIGENVECTORS,C. IE.MAX ABS(B-CAC').
C
      DIMENSION A(1),B(1),C(N,N)
      DO 17 J=1,N
      DO 15 I=1,J
      IJ=I+J*(J-1)/2
      TEMP=0
      L=0
      DO 13 K=1,N
      L=L+K
      TEMP=TEMP+A(L)*C(I,K)*C(J,K)
13  CONTINUE
      ERR=AMAX1(ERR,ABS(B(IJ))-ABS(TEMP))
15  CONTINUE
17  CONTINUE
      RETURN
      END
```



```

SUBROUTINE INTCHG (A,B,C,ID,ID1,ID2,N,NR)
C
C      THIS SUBROUTINE ORDERS THE EIGENVALUES OF A & B
C      ACCORDING TO INCREASING MULTIPLICITY AND INTERCHANGES
C      THE COLUMNS OF THE MATRICES OF EIGENVECTORS U & V
C      CORRESPONDINGLY.
C
      DIMENSION A(N,N),B(N,N),C(N,N),ID(1),ID1(1),ID2(1)
      N1=NR-1
      DO 1 II=1,N
        ID1(II)=II
1    CONTINUE
      DO 11 I=1,N1
        IT=0
        NI=NR-I
        DO 9 J=1,NI
          NJ=NR-J
          K=NJ+1
          IF(ID(NJ).LE.ID(K)) GO TO 9
          K2=0
          DO 2 II=1,K
            K1=K2+1
            K2=K2+ID(II)
2        CONTINUE
          DO 3 II=K1,K2
            ID2(II)=ID1(II)
3        CONTINUE
          MK=ID(NJ)
          K4=K2+1
          DO 5 II=1,MK
            K3=K1-II
            K4=K4-1
            ID1(K4)=ID1(K3)
5        CONTINUE
          DO 7 II=K1,K2
            K3=II-MK
            ID1(K3)=ID2(II)
7        CONTINUE
          IT=ID(K)
          ID(K)=ID(NJ)
          ID(NJ)=IT
9        CONTINUE
          IF(IT.EQ.0) GO TO 13
11       CONTINUE
13      IF(I.EQ.1) GO TO 19
          DO 16 I=1,N
            K=ID1(I)
            KT=ID1(K)
            DO 15 J=1,N
              C(J,I)=A(J,K)

```



```
      A(J,K)=B(J,KT)
15  CONTINUE
16  CONTINUE
      DO 18 I=1,N
      DO 17 J=1,N
      B(I,J)=A(I,J)
      A(I,J)=C(I,J)
17  CONTINUE
18  CONTINUE
19  RETURN
      END
```


CONTROL PROGRAM

```

      DIMENSION A(465),B(465),X(465),U(30,30),V(30,30),
1          M(30,30),Q(30,30),P(30,30),L(30,30,30),
2          PI(30),MM(30),ALPHA(30),BETA(30),GAMM(30)
      DIMENSION FMT(18)
      INTEGER P,PI,ALPHA,BETA,GAMM,Q,T,RHO
      COMMON ZERO,RHO
      REAL L,M

```

```

C
C
C

```

```

      INPUT A & B

```

```

10  READ(5,100,END=50) N
      N1=N*(N+1)/2
      READ(5,101) FMT
      WRITE(6,102)
      READ(5,FMT) (A(I),I=1,N1)
      READ(5,FMT) (B(I),I=1,N1)

```

```

C
C
C

```

```

      INVOKE GRAPH ISOMORPHISM ALGORITHM

```

```

      ZERO=0
      CALL GRAPH1(N,A,B,U,V,Q,M,L,P,PI,MM,ALPHA,BETA,GAMM,X)
      IF(RHO.EQ.0) GO TO 10
      CALL GRAPH2(N,A,B,U,V,Q,M,L,P,PI,MM,ALPHA,BETA,GAMM,X)
      CALL GRAPH3(N,A,B,U,V,Q,M,L,P,PI,MM,ALPHA,BETA,GAMM,X)
      GO TO 10

```

```

50  STOP
100 FORMAT(I10)
101 FORMAT(18A4)
102 FORMAT('1')
      END

```


ISOMORPHIC EXAMPLE

INPUT DATA

```

      4
(8F10.0)
      0.0      1.0      0.0      0.0      1.0      0.0      1.0      0.0
      1.0      0.0
      0.0      1.0      0.0      1.0      0.0      0.0      0.0      1.0
      1.0      0.0

```

COMPUTER OUTPUT

```

      1..      1      2      4      3
      2..      1      3      4      2
      3..      2      1      3      4
      4..      2      4      3      1
      5..      3      1      2      4
      6..      3      4      2      1
      7..      4      2      1      3
      8..      4      3      1      2

```

** NO MORE ISOMORPHISMS **

NON-ISOMORPHIC EXAMPLE

INPUT DATA

9
(8F10.0)

0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0
1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0
0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	1.0	0.0			
0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0
1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0
1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	1.0	0.0			

COMPUTER OUTPUT

..** A AND B ARE NOT ISOMORPHIC **

B29896